Comput Syst Sci & Eng (2015) 6: 21–30

International Journal of
Computer Systems
Science & Engineering

Web-based applications for open display networks: Developers' perspective

Constantin Taivan, Rui José and Bruno Silva

Centro Algoritmi, Campus Azurém, 4800-058 Guimarães, Portugal
E-mail: {constantin, rui}@dsi.uminho.pt, brcpsilva@gmail.com

Open Display Networks represent a new paradigm for large scale networks of public displays that are open to applications and content from third parties. Web technologies may be particularly interesting as a technological framework for third-party application development in Open Display Networks because of their portability and widespread use. However, there are also significant challenges involved that result from the specificities of this particular usage domain and the lack of specific development insights for this context. In this work, we address the concept of public display application (display app) from a development perspective. The overall goal of this paper is to identify and characterize some of the key specificities of display applications and the appropriate Web solutions that can serve in the development of this type of application. The contribution of this paper builds on our extensive experience with the application development for a real world public display infrastructure and also on a short-term experiment with third party developers. Overall, the results show that Web technologies are valuable building blocks for public displays applications and their adoption is not only a subject for adaptation procedures but also for redesigning their use according to the characteristics and user experience offered by public displays. This research will inform the design of new Web-based models of display applications and shed light on the challenges that may impede third party development and the evolution of an application ecosystem in this area.

Keywords: public displays; ubiquitous computing; third party applications, Web technologies, Open Display Networks

1. INTRODUCTION

Public displays can be found in all sorts of urban spaces. They often operate under a content management model in which content is orchestrated at a central location and then distributed to the displays just for presentation. Open Display Networks represent an alternative model in which large scale networks of public displays are open to applications and content from many sources [1]. This new model creates the opportunity for third parties to create and publish content in the form of applications to be used at any display across multiple administrative domains, promoting openness as a source of value for all the parties involved. Multiple entities anywhere in a global network could become co-creators of value by developing new applications in which new ideas could quickly be shared and feed the innovation cycle.

The overall vision of applications [2] for Open Display Networks requires addressing a broad range of challenges. This includes the ability of applications to be globally available so that they can be used anywhere across a global and open network of public displays. Those applications would also have to be able to run across many different types of displays and execution environments with very heterogeneous sets of computing platforms, interaction techniques, display sizes and form factors. While an application must be globally available, they also need to exhibit a specific situated behavior at each of the many domains in which it may be in use, similar to what we have seen in mobile computing environments [3]. The content these applications would employ is to be managed and exposed in a way to optimize its use in public displays. Finally, at each domain, many applications may be available and would be concurrently

used by multiple users with interfaces that can distributed across an eco-system of large and personal displays – such as mobile phones.

In this study, we specifically address the use of web technologies as the technological framework for third-party applications development and deployment. Web technologies can be particularly valuable in regard to openness, portability, widespread availability and easy to deploy in large scale [4]. The main benefits lay in the wide usage of web technologies and their ability to be supported across many platforms. A vast range of tools already exist and many people already have the competences to create all sorts of web content. This includes also the emergence of new standards and specifications such as HTML5 and CSS3 which make possible dealing with the requirements imposed by Open Display Networks.

The main limitation would be that web engines were not conceived for this purpose as they were designed based on different usage assumptions. For instance, a web browser is mainly used for rendering content and there is no interaction with the web content (web sites, web apps) being rendered. A proper use of web technologies should enable web content to optimize its presentation in collaboration with the container, i.e., a web browser or a specific web-based player, where it is being presented, and it should provide a framework for extending the scenarios and functionalities of the containers. Moreover, web-based applications face limitations in the access to device specific resources (e.g., RFID, sensors). In particular, the use of Web technologies in display systems poses many new challenges and simply showing normal web pages makes for poor signage content [5]. While the ability to present web content from a specific URL is not a challenge in itself and is already an integral part of almost any display system, the overall context of how this content is selected, obtained and adapted to the circumstances of a particular display is something that is not well matched by prevailing web application solutions.

There are other alternative ways to approach the creation of display apps, e.g. virtual machines [6] or cloudlets [7], and we are not claiming that Web technologies would necessarily be the best approach for all scenarios. We just considered that Web technologies would be the right context to study the requirements of applications and then, based on emerging limitations, identify more clearly the situations in which alternative technological frameworks could be adopted. For example, it may be needed, depending on the requirements, to consider specific software components that would complement Web technologies such as the usage of display-specific virtual machines to improve robustness against server and network problems [6]. However, in this work we will only focus on the use of Web technologies.

In this paper, we study the key specificities of using Web-based applications as the primary driver for the experience offered by public displays. The research is focused on the application development perspective and not so much on the goals of the applications. The study is grounded on our experience in creating a varied set of Web-based applications for public displays and also on interviews with third-party developers that have created displays applications. The combination of these diverse perspectives has enabled us to consolidate the many challenges that developers need to consider when creating web applications for the execution environment of public displays. Overall, the results show that Web technologies are valuable building blocks for public displays applications and their adoption is not only a subject for adaptation procedures, but also for redesigning their use according to the characteristics and user experience of public displays. These results inform the repurposing of Web technologies as an appropriate technological background for display applications.

2. RELATED WORK

Our research builds on the analogy of using Web technologies for mobile devices. W3C developed a number of technologies that explicitly address the specificities of mobile devices (e.g. network costs and delays, memory and CPU limitations, input differences, context-aware capabilities): CSS Mobile, SVG Tiny and XHTML for Mobile [8]. A similar process occurs with the use of Web technologies in TV sets, also addressed by W3C [9]. With the emergence of IP-based TV devices, a.k.a. connected TVs or Smart TVs, Web applications can also be made available in TV sets.

Building on the mobile app store metaphor, Clinch et al. [10] present a set of design considerations for app stores for public displays. Conceiving such application stores faces specific challenges when compared with the mobile counterparts, such as dealing with multiple stakeholders, new business models and scheduling requirements. While mobile application landscape includes well established platforms and ecosystems for running, developing, distributing and selling of applications e.g., Windows, iOS, Android, for public displays there is no product or any established system that enables the creation and everyday usage of display applications. Although the research community is increasingly trying to offer insights on how display applications should look like, the support for third party development in the context of public displays is not reached yet.

Various display prototypes used Web for their infrastructure and applications [11, 12, 13, 14]. The simple inclusion of an application URL is seen as a regular pattern to provide web content or interactive services to a public display. In particular, Social Networking Services (SNAs) are considered as a dynamic user contributed content source that can add more value for public displays [15, 16]. The integration of SNAs is also explored by Locamoda – a company that provides several place-based social media display applications focusing on enabling personalized and interactive experience with display content [17].

Very often public display installations are conceived as distributed applications and common design goals include ease of deployment and content creation, maintainability and robustness. A reference example is the display infrastructure in Oulu [18], with 12 interactive displays that support the deployment of services in form of web-based applications. Oulu's multi-application public displays based its design on the Web paradigm and enable content contribution from multiple third parties. Services may reside anywhere in the Internet under a simple URL. Their experiences over a period of three years have shown the many specificities of public displays, which mainly result from the public context of this type of installations. In their work [6], an approach based on virtual machines and web technologies was suggested as an appropriate model for supporting applica-

tion deployment. e-Campus public display infrastructure from Lancaster University [19] is another relevant example for using web applications as means to personalize user experience in front of large displays [20]. Based on a mobile Android application users can locate the nearby displays and configure what content to see as part of the associated display web applications. Memarovic et al. [21] identified a number of challenges when moving from personalized Web content to personalized content for public displays including user identification, profile location, profile content, content tailoring, model refinement and applications that require personalization. In their vision, public display networks require novel approaches for personalization and existing web personalization solutions cannot be used as they are employed in desktop computing environments. Overall, the Web and its set of enabling technologies are attractive for building displays infrastructures and applications but not much is known about the specificities of display applications and the implications they might have on these technologies.

Our previous work focused on understanding what is an application for the context of public displays and, in particular, what are the opportunities and limitations of Web as a technological framework. In a short-term study on creating web-based display applications by third party developers [22] we found that developers face a set of challenges such as visual adaptation, managing of content and fault tolerance support. In order to effectively leverage on developers' web experience, the study uncovered that two conditions are essential: 1) provide a clear description on the specificities of display applications and 2) provide appropriate tools to facilitate the development process. To better highlight the specificities of display applications and their implications on Web technologies, based on our long-term developing experience in creating this type of applications, we elaborated a set of four key issues that web applications need to consider when repurposed for the usage in public displays: *content management, content addressability, visual adaptation and integration with the execution environment* [23].

While the research community is already working on applications for public displays, this paper is the first to address the concept of display application from a development perspective. From considering the extension of current web development practices and expertise to support the creation of web-based display application to a detailed journey into the long-term development processes, this paper aims to identify what makes a display application different from its desktop and mobile counterparts. Driven by the vision of applications [2] for Open Display Networks, and not by a platform or system specific incentive (which might limit the designs, perspectives and features), our goal is to reach a generic understanding about the specificities of display applications that can frame the development of many different types of web-based applications across an unknown and diverse set of multi-application displays.

3. WEB APPLICATIONS FOR PUBLIC DISPLAYS

In this section, we clarify our main assumptions about Web-based applications for public displays. For the purpose of this work, we consider a display application to be a Web-based application whose primary goal is to render content on a public display. Like any other Web application, display applications are based on Web technologies and standards, e.g., HTML, JavaScript and CSS. Display applications are shown in full-screen and run on standard Web engines or other types of specially tailored Web stacks and they encapsulate both content and the means to render that content on screens. Deploying applications in public displays require a scheduler component which in our case is a software player that actually runs the application within the browser. Our current version of the player[1] uses Internet Explorer browser and platform specific libraries in order to control when applications should start and, how long they should present content. The need for supporting disconnected operation and specially tailored content management policies, led us to assume a rich client model in which the core of the application is running on the display node. Each application will have its own JavaScript code to handle, on the display side, issues such as obtaining and managing the content items that the application will need, caching and prefetching of content, or dealing with network disconnections.

We also assume that these applications entail a clear separation between content creators and particular displays, reflecting the need to develop applications that may potentially be used anywhere. Therefore, applications must be developed without any assumptions about their execution contexts. This implies dealing with the potentially strong variations in the resources that may be available across locations. Portability, in the sense of being able to work across multiple display platforms, it is the most obvious requirement, but there is also a need to accommodate other differences in the operational environment, e.g., display sizes or interaction modalities, as well as variations in the associated information space.

Regarding the user interaction model, we assume that multiple display viewers can interact with the applications only through mobile phones. Our software infrastructure enables viewers to personalize some of the content of the applications but do not allow them to influence the application presentation times or change the application schedule. Within our display infrastructure, these tasks are handled exclusively by display owners. Overall, the model of application presentation is driven by the content sliding by with a fixed interval for each application and without any direct interaction from viewers such as using touch or gestures, the only possible interaction being remote through the usage of mobile devices.

3.1 Application Development

As part of our work in Instant Places [24], a Web-centric platform for place-based screen media, we have specified a model for Web-based display applications and developed a considerable range of applications that were deployed in our prototype displays. While we are not claiming that our development approach would be the only possible solution for Web-based display ap-

[1]The rationale behind this player is not within the focus of this paper. It represents the main approach for application presentation used in Instant Places display infrastructure [24]. However, in this work we provide insights into a new type of player that is based on Chrome browser and helps addressing the requirements of display applications without the need of any platform specific components (Section 6.1 will describe our arguments for the new player).

plications, this is a perspective that has evolved over the years with our ongoing research in this topic.

Our applications represent a diverse set of requirements and have allowed us to gain a broader understanding of the use of Web technologies in creating them. The most significant applications are: *Video* app shows YouTube videos and allows users interactions from their mobile devices in forms of comments and likes; *Posters* app presents multimedia posters published by visitors which were approved by the place owners; *Places* app – aggregates the contents of a place such as pins, checkins and has three views: a) check-ins and the current pins – shows the name of the place, recent checkins (for one week) and the place pins; b) Facebook account of the place by using a given id one can see the related posts; c) Facebook albums – allows to present the place Facebook albums; *Pins* app shows all the pins from the place such as users driven pins from the checkins and the place specific pins; *Dropbox* app allowing place owners to present files from a Dropbox folder. *Facebook* app for showing content from selected Facebook page walls; *Twitter* app that shows discussions posted to a specified hashtag; *RSS Feeds* app for showing selected news feeds; and *Media RSS feed* app for showing the feeds from a media RSS, targeted for images and videos. The applications developed have all been made available across multiple deployments where they have been used by local communities on a regular basis. These testbeds have been pivotal in enabling us to assess a more diverse set of requirements and contextual assumptions.

3.2 Development Tools

We created a developers' Web site with key information on how to develop these apps and also with the following set of development tools: *Application Generator*, *Instant Places library* and *Media Simulator*. The *Application Generator* provides developers with the possibility to generate a ready-made application structure. This considerably reduces the initial development effort and it promotes the use of patterns and components that are known to work better with this type of application. This was achieved by the generation of a Hello World display application, which constituted the skeleton for the creation of other applications. The *Instant Places library* provides an abstraction layer for the Instant Places service that enables applications to integrate dynamic data into their content, more specifically place-based information about their surrounding settings, i.e. sensing and interaction information associated with displays. Finally, the *Media Simulator* allows display applications to be tested in their target execution environment, i.e., display nodes' player that uses Internet Explorer browser. Instead of deploying applications to the real display infrastructure, developers have the ability to use this tool to check in advance if a display application is ready to be shown on a public display. Based on a set of guiding reference tests, e.g., resizing the window of the application, unplug the network cable, a developer could observe the behavior of the application.

In addition to these tools, we also provided developers with a few additional guidelines on how to handle key issues such as network disconnection and visual adaptation. Building a fault-tolerant application is essential to public display environments, because we do not have an end-user that is ready to solve the problem. We included a set of code blocks for the cases when no data was fetched or it took too much time to show up, e.g., splash screens routines for masking application startup delays or show something to its audience while external data is being fetched. For example, the Hello World application generated by the Application Generator already included a splash screen hiding the error of no connectivity. To handle the diverse resolutions and orientations that public displays can have, there is a need to employ at least some basic techniques for making the application content look good and – especially – readable. Our initial Hello World app already included a technique based on CSS media queries. It allows developers to add expressions to media type to check for certain conditions and apply different style sheets. For example, one can have one style sheet for large displays and a different style sheet specifically for mobile devices. The technique is really helpful because it allows adjusting to different resolutions and devices without changing the content. The condition that is often verified to trigger the changes is the viewport width. When the viewport is too narrow, applications can adjust the font and some box sizes.

4. METHODOLOGY

In order to investigate the key specificities of using Web-based applications as the primary driver for the experience offered by public displays we conceived a methodology anchored on three main activities: 1) an application hackathon to introduce new developers to this type of applications and assess the effectiveness of our development tools; 2) an internal focus group about our experience in creating a varied set of Web-based applications for public displays; and 3) interviews with third-parties developers that have created and deployed applications based on our approach. The combination of these diverse perspectives has enabled us to consolidate the many challenges that developers need to consider when creating Web-based applications for the execution environment of public displays.

4.1 Application Hackathon

We conducted a short-term development experiment in which we investigated the learn ability and usefulness of our development tools by new developers. The assessment of our development tools was achieved by adopting an *informal and controlled laboratory evaluation* [25][26]. We invited five participants to create a given display Web application by using our guidelines and tools and interviewed them about their overall development experience. All of them had basic Web development skills, e.g., JavaScript, HTML and CSS, and had never built a display Web application.

A week before the experiment, we sent participants the URL of the development Web service so that they could learn the basics of the process. At the beginning of the experiment we gave them a brief tutorial of about 10 minutes in which we introduced the concept of display application and explained the APIs. They were then asked to build a new display application, i.e., a poster grid app, based on the Hello World example. To do this, we

formulated three development tasks that led developers to create the given app. The first task was to put the Hello World app running and test its execution. For this, they needed to install the App Generator and output an application example and Media Simulator for being able to test it. The second task was to use the Instant Places library for getting place related data, such as the *place name*, *place image* and *posters*. Finally, participants were asked to show the posters in a grid by using some CSS rules. In this step, developers needed to use splash screens and configure them to last for at least 3 seconds; support fault tolerance functionality (lack of data, lack of connectivity); prepare the app to be displayed correctly in an iPad or in another device of similar dimensions and test the application using a desktop web browser and Media Simulator tool. Throughout the experiment, participants were encouraged to raise questions and they had four hours to complete all the tasks. At the end, each of them was interviewed about their experiences with our development tools. The interviews were audio recorded and the code produced by developers was kept for subsequent analysis.

4.2 Internal Focus Group

To gain a better understanding into the concerns of application development we conducted one internal focus group with three participants. One of the participants was the main developer of the Instant Places platform and the rest of two were researchers. The focus group was employed as an unstructured discussion around the key aspects of display application development using Web technologies. The emergent ideas and aspects were then prioritized and organized in four themes: *content management, content addressability, visual adaptation and integration of applications with the execution environment* (see [23] for a detailed view). This constituted the analysis structure for the interviews with third-party developers.

4.3 Interview with Third-Party Developers

Following the key topics outlined in the focus group activity (and reported in ....[23]) we conducted three interviews with third party developers in order to get insights into the development experiences with display applications. The interviews included three developers that created several real applications based on our approach and guidelines. Developers did not participate in the specification of our model for display application. The applications were conceived to be deployed in our display infrastructure and to be used on daily basis. The main goal of the interviews was to understand to what extent developers addressed the issues of *content management, content addressability, visual adaptation and integration of applications with the execution environment*. As well, we looked to understand the opportunities and limitations of Web technologies as the technological framework for the creation of this type of application.

5. DEVELOPMENT EXPERIENCE

This section highlights the main findings in regard to the development process experienced by new Web developers that, as well as third party developers did not take part in the specification of our approach for creating display applications. The overall view of this experiment was positive, even for less skilled developers. All the participants have achieved the key development goals without wasting too much time in writing the code. They found our tools useful and necessary for the first contact with Web-based display applications.

Participants had some initial effort to grasp the specific concepts associated with displays applications, but after that, they were quickly able to master the process. Developers could easily follow the documentation provided by our development Web site. Even though this was optional, all participants used the Hello World app generated by the App Generator tool as a template to start implementing the new display application. The participants didn't think very much about the structure of the application, which meant that the use of Application Generator was effective. When we asked developers how it would be to develop without this tool all of them responded that would it be difficult or even very difficult.

Developers had enthusiasm for this experiment despite their weaker experience with some of the required Web technologies. This is demonstrated by the fact that all of them succeeded in applying their Web development skills to develop a display app. However, a few of them experienced difficulties in understanding and using all specific development and testing scenarios, e.g., implementing the splash screens or providing the required code blocks for a fault tolerant display app. Only one of them could entirely test the app execution behavior.

Due to the fact that our display application was not too complex, it just required a set of API requests, the code source is quite identic among the participants and the final applications share the same structure and very similar lines of code. Having a previously scaffold app structure proved to be comfortable to the participants and reduced the amount of code they had to write. Developers ended up not writing much code and not changing the application structure at all. Instead, their effort was mostly to combine various code blocks and configuring them appropriately. However, one student noted that the integration of our code blocks was straightforward, while making various customizations was not so easy.

Using the Instant Places API library was something that proved to be very handy. Although there were some initial problems in understanding the meaning of our API and the related code blocks, after getting the *place name*, they easily succeeded to get further data, such as *posters*.

Developers had difficulties when testing their apps because they weren't familiar with any tools to accomplish this task, e.g. Fiddler. Most tests were made using a common Web browser while the Media Simulator tool was just periodically used to rule out eventual errors related to the different Web engine of display players. Only one student did not test at all the new application execution, neither in desktop Web browser nor in Media Simulator tool. The others tested the application but encountered various difficulties.

Participants were really motivated by the innovative field of

usage of Web-based applications and recognized their big potential when deployed in real world settings. They associated display apps with mechanisms to publish content, such as replacing the traditional paper based posters with digital forms of content. In their final comments, they all referred particular features for display Web apps, e.g., a display app should provide content that is dynamic, personalized and place-based.

6. IMPLICATIONS FOR WEB TECHNOLOGIES

This section consolidates our findings on the identification of the key challenges that Web applications need to consider when being repurposed for usage in public displays. These results are mainly informed by the long-development experience of third-party developers. The discussion is organized around four themes: *content management*, *content addressability*, *visual adaptation* and *integration with the execution environment*. For each of these themes, we identify the specific challenges regarding the ability of Web technologies to support display applications requirements and how we approached them so far.

6.1 Content Management

In traditional interactive Web browsing, content selection is assumed to be under the control of a single user, who may at any moment request a new content resource or be prompted to provide any necessary data, including, if needed, authentication data. In a public display system, content presentation can be mainly autonomously determined by the system itself, which must be able to guarantee that any necessary configurations or content selection options must have been done before the display starts presenting content. In the following we present three key specificities regarding content management issue.

Avoiding idle times caused by fetching content from servers. Idle times may not always be bad, but on a public display people expect the same smoothness and performance in content presentation that they are used to see in traditional television broadcasting and even in other existing display systems. Therefore, loading time should never correspond to idle presentation time. If the system stops presenting content while the next content is being loaded, the user experience is completely destroyed.

Our first approach in dealing with this issue, which is employed in most of our applications, is based on *splash screens*. A splash screen is a kind of animation that informs users about the current application being loaded. While this approach is neither specific for Web technologies nor for Web-based display applications, it provides an effective way to notify the users about what is going on. However, it has the drawback that if the idle times increase too much users would see the same splash screen excessively. A second approach in dealing with the idle times is based on a custom made pipelining technique called *in-app prefetch* employed by a set of applications called schedulers. The schedulers are able to present in full screen other apps (modules). For instance, a scheduler puts a module running while prefetching the next one. The prefetch of a module is implemented as a hidden request and as soon as that module is ready it triggers an event that is used by the scheduler to activate the next module.

We are currently working for a better approach that would be based on a new type of player that will exclusively be based on Chrome browser without any additional and platform specific components. Such a player will be responsible to prefetch, run the applications and coordinate them. For example, when the app will be ready it will inform the player that it is able to run and the player will act accordingly.

The decision for choosing Chrome was informed by our attempts to find a browser that can run a Web-based player that will run our display applications in a way that allows overlapping operations like prefetching content while presenting other applications. Moreover, Chrome browser offers support for Cross-origin resource sharing (CORS) technique – which is an essential feature of the applications within our development model. The iframe approach has the drawback that when the content from an iframe blocks or it does not respond (e.g., Javascript errors), the others blocks as well. Our solution to overcome this issue is based on developing a Chrome App[2] that can employ *webviews* instead of *iframes*. A *webview*[3] is a way to actively load live content from the web over the network and embed it in a Chrome app. The advantage is that a webview runs in a separate process than the main app and it doesn't have the same permissions as the main app and all interactions between the Chrome app and embedded content will be asynchronous. In this way, the main application is kept safe from the embedded content. Since the current player is system-dependent, the new player that we are working for Google Chrome browser will also alleviate the restrictions associated with the portability and deployment of our display infrastructure software.

Make any content fetching errors transparent to users. In the traditional Web browsing experience, when a content resource cannot be obtained, the result is a message error notifying the user about the problem and possibly giving additional indications on how to proceed. On a public display, content loading errors should never result in error messages being shown, because those people who would see the message might have not requested the content and probably cannot act to solve the problem. This means that applications must be able to catch any such errors before they show up infamously on the screen, and report them through some alternative channels so that appropriate corrective action can be taken. It also means that a fallback strategy must be in place to be activated whenever a content loading error is detected. The application itself may have the ability to detect the problem and present an alternative content that is available.

In our applications we tried to catch and hide any errors and inform users about those resulted from content loading processes. Currently, we use a very naïve approach that displays a funny custom message in form of a splash screen that redirects users to the main Web page of the system. Since our current player does not have the possibility to switch between applications, users will be promoted with the same screen message for an extended amount of time.

However, this is not the intended approach and we are considering a better solution by developing of the new player (previously introduced) that will be able to solve this issue. The

[2] https://developer.chrome.com/apps/about_apps

[3] https://developer.chrome.com/apps/tags/webview

current app may inform the player to show other app because its content is not ready or show another splash screen, or the player can detect that the app is not ready and skip it from presentation. In general, to avoid any error messages, the app should recognize the problem before diving to it. This should also eliminate any idle times.

Support disconnected operation. In a personal browsing scenario, disconnected operation is not normally very relevant. Either it would be limited to content already seen by the user or the system would have to be able to anticipate the intended content. In public displays, where content is often designed around content loops, cycling through the same content multiple times may even be seen as the expected behavior. The ability to maintain a normal or slightly deprecated operation when disconnected is thus essential.

Disconnected operation is currently the major problem we faced in our applications. Currently, we are using a solution based on 3 technologies: App Cache [27], HTML5 Local Storage [28] and IndexedDB [29]. Overall, a key aspect in offline behavior is to considering the frequency of data updates. Most of our apps may survive a few hours of disconnection. For instance, *Place* app is more dynamic and requires more frequent data updates in order to show relevant information; if there is no network we lose the presences information. A critical situation would be if an app does not have any content to show and in this case it should inform the player which might schedule other applications, some predefined content items or apps that do not change too frequently such as those with 2-3 updates a day.

In regard to local storage techniques, our current work focuses on implementing a functionality to store images locally. We have just started to develop a wrapper around IndexedDB which will allow us to increase the local storage size to GB instead of MB used in HTML5 Local Storage mechanism. Right now, the only issue is that the development process is a bit more complex and harder to develop these apps because we have to store everything we have in the app even the content that comes from external servers. In particular, the challenge is that the images from external servers do not allow cross domain requests and we cannot retrieve the images for storing locally. A solution might be to use our servers to proxy them trying to retrieve their content.

In conclusion, our solution for disconnected operation may cover the case with 2-3 updates per day and in this regard HTML5 Local Storage and App Cache work pretty well. For the case of applications that needs data within seconds we may have a problem. Besides the local storage limits, we are also limited by the lack of available services that tackle this problem for desktop Web apps.

6.2 Content Addressability

A key distinction between a normal Web app and a display app is that in the latter there is a much stronger need to systematically handle the data exposed by the application. In a normal user-driven browsing scenario, the issue is mainly about links and navigation menus that the user will invoke as needed. When the content is being consumed by a display system, the issue is mainly about exposing and characterizing the set of content items available in the application to allow the display system to integrate that content into the local content schedule. Exposing content as atomic presentation units, i.e., the smallest segment of content that can be presented on a display, is not mandatory but is crucial not only for automated scheduling purposes, but also because it can become the enabling element for many other key features such as cache management, prefetching, auditing, logging, scheduling and social interactions around content.

In public displays, presentation units might play a similar role as the concept of permalink in blog posts. For instance in a blog scenario, with the emergence of permalinks (permanent links) posts can now have a specific URL that remains the same even when they are no longer visible in the blog front-page. This permanence of the links enables those posts to be linked by other sites and provide a reference that supports many other key web functions, such as searching, traffic measure and comments.

In our approach, we mainly considered one scenario in which applications can expose their resources. Our system enables apps to expose resources and they can do it by exposing a URL to the resource. For instance, people may interact with application resources by getting their URLs in personal mobile devices. Then, users can have a closer look to a specific content item regardless the content shown on public displays. In this scenario, it is up to applications to decide which resources to expose, if any at all. For instance, Video app has this feature and the system allowed the content of the screen to be posted on people's mobile phones so that they could subsequently access the respective application resources independently. Other possible scenario not covered by our work, would be that applications to be able to expose their resources in order that other applications could integrate them in different ways. For example, there might exists applications that aggregates content from many other applications and offer an integrated experience to the user such as a dashboard app that provide a content overview of the applications running in a display from a given place.

6.3 Visual Adaptation

We call visual adaptation the process of adjusting the content appearance of a Web site or Web app to the browser screen dimensions. For instance, visual adaptation is performed when a desktop Web site adapts its text font size to be legible in a smartphone. While this need for visual adaptation is common in desktop and mobile Web usage, the adaptability range in public displays can be much more extreme and the role that users can have in assisting the adaptation process is more limited. In public displays, there is much more uncertainty about possible displays sizes and properties. Content may need to be rendered on small displays or small regions of a large display, but it may also have to fill an entire display wall. Additionally, the position of the display in regard to viewers may also face dramatic variations in distance that will severely affect the adequate visualization of content.

For all the applications we are using responsive web design within a limited range of display resolutions or screen sizes. Our approach is mainly based on using percentages instead of fixed sizes for any visual elements in the application. We also used the *media* tags on the CSS that can control the sizes of

elements in relation with the screen size. For example, for a given screen width we are going to give a certain width to an element. While CSS media queries served very well for implementing the responsive web design, we reached a limitation related to the sizes of images and the text length that came from Twitter or Facebook APIs. The first assumption in our applications was that people cannot scroll text or images on the screen, so, all the content need to fit a single display. Due to the lack support in availably plugins[4], we developed our text adaptor plugin that can enlarge the text within a given min and max font size and ellipsis the rest. Thus, the drawback of our approach was to cut the text that did not fit the screen. As well, handling images involved some manual configuration depending of the availability of the image properties.

A second factor that influenced our adaptation solution was the viewing distance. For the text, the approach was to experiment with different font sizes in a way that allow people to get the content from a few meters away. In particular, the Video app highlighted that visual adaption may be different depending of the users engagement with the app. Users could see the videos on their mobile devices so, if they are in the back of a room this might not affect too much. They can come closer or just watch the videos on their mobile phones. Therefore, the viewing distance is much important for the first phases of the engagement. Since the users are already joined with the display app they may walk away from the display. In other words, visual adaptation is an interesting factor that may influence the displays' role to entice for interaction and to join the system. For example, in a big room an application may adapt the content (providing only keywords) in a way that people at a large distance can notice what is going on and manifest interest in joining the system, while for small rooms it can further provide additional content having the same goal to entice people to get in. In conclusion, our experience shows that responsive web design technique could not only consider the screen size but also the space properties, e.g., maximum available viewing distance, in order to better adapt its content and entice interaction.

A third factor with a significant impact over the entire public display experience was the visual aspect of the applications. Developers struggled to design applications that do not have the look and feel of a Web app or Web site. Instead, our applications need to present content in a natural way so passersby could get that content easily understood. Developers tried not to employ as many elements as they are present in desktop Web-based applications. For instance, we did not use columns and menus as our particular model for user interactions assume that people interact with display applications through their personal mobile phones.

6.4 Integration with Execution Environment

An underlying assumption behind the notion that displays will be open to many applications from third-parties is the idea that any particular application is expected to be one of many that may simultaneously be running on a single display and requires sharing the display resources, e.g., screen real estate or interaction features. This means that a display system will employ optimization protocols between applications themselves and between applications and their execution environment. Application developers do not know a priori the conditions in which their apps will be running. Thus, applications could use these protocols to coordinate between themselves to exhibit an integrated behavior, e.g., avoiding contradictory presentation times. In the following we present three key aspects regarding the integration with execution environment.

Communication between the player and apps and between apps themselves. This type of communication may help to coordinate the content scheduling process, by allowing the container to inform the app about the best moment to start, stop or prefetch content presentation, and also inform the app about the allocated presentation time. Likewise, the app may inform the container about internal events that are relevant for the scheduling process, such as content loaded or interactions received from users, or it may request additional presentation time, request to be removed from presentation or even take the initiative to request presentation when certain events occur. In order to optimize network resources usage, display applications should report their possible errors to the execution environment, so that it can channel them more efficiently, e.g., to some application quality service that then informs developers. Ideally, developers should have access to libraries and tools for capturing errors and channeling them appropriately.

Since our new player is not yet ready we did not address the issue of integration with it. This integration will be a great plus for the user experience offer by public displays in general. The main role of the player is to act as a controller that is responsible for the execution of each application. For instance, the communication between applications and player is needed in order for the apps to inform the player when they are ready to be presented, i.e., control the application start/stop time. A further example for this communication includes managing of application errors, e.g., reschedule an application or remove it from the presentation. In particular for the Video app, if there is a video currently playing, instead of stopping the video when the time allocated to an application elapses (actual behavior) the application can ask for more presentation time.

Accessing local resources. The optimization protocols between application and execution environment could also allow apps to have access to local machine resources, e.g. a camera/Kinect device, or obtain information about the environment, e.g., display ID or presence of people in the vicinity of a display.

So far, our applications did not consider the need for accessing local machine resources. While this topic is of interest for us, our focus was the use of Web technologies to study the requirements of applications for public displays. For accessing the hardware resources, the usage of third-party libraries would be mandatory. DepthJS[5] is one of such library that is under development of MIT Media Lab. It is an open-source browser extension and plugin (currently working for Chrome) and allows any Web page to interact with the Microsoft Kinect using JavaScript. DepthJS provides the low-level raw access to the Kinect as well as high-level hand gesture events to simplify development.

Keeping state. Most of our applications keep state between subsequent application calls. The applications store the state of

[4] http://simplefocus.com/flowtype/ (just one example)

[5] http://depthjs.media.mit.edu – Last accessed on 7.07.2014

the last content item shown and next time when it plays will not show the same item again. In the majority of the applications we used HTML5 Local Storage without any specific considerations (only Video app used cookies).

7. FUTURE WORK

Deploying a display infrastructure in a real world environment requires many software components to be in place. One such component that has a key role in the overall experience of public displays is the player that runs the applications and is able to act as a scheduler. As Open Display Networks assume the provision of applications from many sources that can be scheduled in different ways, e.g., as users pass by, an app is not ready, the display system would need to take decisions dynamically and most of this functionality can be embed in the player. In our approach, we considered the player as a Web-based component, that runs in a browser and coordinate the most of the application events. Therefore, the first priority for the future is to have the new player ready and embed the features such as communication with the apps using HTML5 Web Messaging [30], remove or add applications to the schedule, log the errors and report them to the application developers. Having this player in place will enable a more fluid application presentation without putting people to wait too much for the application to be ready. A second priority is the communication between our APIs and apps. Based on Web Sockets[6], applications will be able to show the new content as soon it arrives and it is ready rather than refreshing at 30s. And, a third priority is to get a better understanding of disconnected scenarios for display applications and to improve our knowledge on the available technologies like App Cache and IndexedDB, despite the lack of current services that tackle these issues in desktop or mobile computing landscapes.

8. CONCLUSIONS

The openness and portability of Web technologies are key properties when considering the development and usage of third-party applications in Open Display Networks. However, public displays represent a new frontier for Web technologies, with novel usage situations and technical requirements. This means that a Web-based development model for display applications would be informed by specific activities e.g., adaptation procedures, creation of new tools, complex configuring, redesign that make sense only for the context of shared, large pervasive displays and are not relevant for desktop or mobile computing cases. For instance, not having a scroll in a public display is something that makes different the development approach which might lead to the emergence of new techniques. Similarly to what has happened in the mobile landscape, there is a need for specific approaches that enable display applications to seamlessly integrate the content they generate on the presentation context of public displays.

As part of an urban deployment of public displays, we have created and deployed a number of applications, with different characteristics and requirements. Based on these experiences, we have consolidated our view on the best ways to adapt Web technologies for the creation of display applications. We highlighted that while the Web has various building blocks that can serve our scope, display apps have a number of specificities with important implications on how web technologies can be used in this context.

Firstly, we found that in order to effectively leverage on developers' web development experience, clear development specifications, guidelines and tools are required for creating Web-based display apps. Secondly, we provide a better understanding of some of the specificities of display apps in form of a set of design and engineering issues that challenge developers, and how it may shape the emergence of new Web-based models and wide available application ecosystems. So far, we learnt from what we have experimented with these specificities that: while many Web techniques are ready available for the adoption for this type of application, the challenges arise from the particular usage scenarios and user experience offered by Open Display Networks.

[6]https://www.websocket.org/

Acknowledgements

The research has received funding from "Fundação para a Ciência e a Tecnologia", under the research grant SFRH/BD/75868/2011. We would like to thank our third party developers João Teixeira, João Casal and Maximilian Müller for their valuable insights.

REFERENCES

1. N. Davies, M. Langheinrich, R. José, and A. Schmidt, "Open Display Networks: A Communications Medium for the 21st Century," *IEEE Computer*, pp. 58–64, May-2012.
2. C. Taivan and R. José, "An application framework for open application development and distribution in pervasive display networks," in *Proceedings of the 2011th Confederated international conference on On the move to meaningful internet systems, Berlin, Heidelberg*, 2011, pp. 21–25.
3. D. Martín, D. López-de-Ipiña, A. Alzua-Sorzabal, C. Lamsfus, and E. Torres-Manzanera, "A methodology and a web platform for the collaborative development of context-aware systems.," *Sensors*, vol. 13, no. 5, pp. 6032–6053, 2013.
4. V. Pawan, *Web Applications Design Patterns*. Morgan Kaufmann, 2009, p. 448.
5. S. Clinch, N. Davies, A. Friday, and C. Efstratiou, "Reflections on the long-term use of an experimental digital signage system," in *Proceedings of UbiComp11*, ACM, 2011.
6. T. Lindén, T. Heikkinen, V. Kostakos, D. Ferreira, and T. Ojala, "Towards multi-application public interactive displays," in *Proceedings of the 2012 International Symposium on Pervasive Displays - PerDis '12, 2012, New York, NY, USA, ACM*.
7. S. Clinch, J. Harkes, A. Friday, N. Davies, and M. Satyanarayanan, "How Close is Close Enough?? Understanding the Role of Cloudlets in Supporting Display Appropriation by Mobile Users," in *Proceedings PerCom'12*, 2012.
8. W3C, "Mobile Web." [Online]. Available: http://www.w3.org/standards/webdesign/mobilweb. [Accessed: 08-Jul-2014].
9. W3C, "Web and TV Interest Group." [Online]. Available: http://www.w3.org/standards/webofdevices/tv. [Accessed: 8-Jul-2014].

10. S. Clinch, N. Davies, T. Kubitza, and A. Schmidt, "Designing application stores for public display networks," in *Proceedings of the 1st International Symposium on Pervasive Displays - PerDis '12*, 2012, pp. 1–6.
11. A. Erbad, M. Blackstock, A. Friday, R. Lea, and J. Al-Muhtadi, "MAGIC Broker: A Middleware Toolkit for Interactive Public Displays," in *PERCOM '08 Proceedings of the 2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications*, 2008, pp. 509–514.
12. N. Memarovic, I. Elhart, and M. Langheinrich, "FunSquare: First Experiences with Autopoiesic Content," in *Proceedings of the 10th International Conference on Mobile and Ubiquitous Multimedia - MUM '11*, 2011, pp. 175–184.
13. F. Alt, T. Kubitza, D. Bial, F. Zaidan, M. Ortel, B. Zurmaar, T. Lewen, A. S. Shirazi, and A. Schmidt, "Digifieds: insights into deploying digital public notice areas in the wild," in *Proceedings of the 10th International Conference on Mobile and Ubiquitous Multimedia - MUM '11*, 2011, pp. 165–174.
14. M. Geel, D. Huguenin, and M. C. Norrie, "PresiShare?: Opportunistic Sharing and Presentation of Content Using Public Displays and QR Codes," in *Proceedings of the 2013 International Symposium on Pervasive Displays - PerDis '13, 2013, New York, NY, USA, ACM*.
15. S. Hosio, H. Kukka, M. Jurmu, T. Ojala, and J. Riekki, "Enhancing interactive public displays with social networking services," in *Proceedings of the 9h International Conference on Mobile and Ubiquitous Multimedia - MUM '10*, 2010, pp. 23:1–23:9.
16. I. Elhart, "WE-BAT?: Web Based Application Template for Networked Public Display Applications that Show User Contributed Content," in *Poster at PerDis 13*, 2013.
17. Locamoda, "Fifteen Seconds or More. Engaging Audiences With Place-Based Social Media," in *White Paper*, 2010.
18. T. Ojala, V. Kostakos, H. Kukka, T. Heikkinen, M. Jurmu, S. Hosio, F. Kruger, and D. Zanni, "Multipurpose interactive public displays in the wild: Three years later," *IEEE Comput.*, pp. 42–49, 2012.
19. A. Friday, N. Davies, and C. Efstratiou, "Reflections on Long-Term Experiments with Public Displays," *Computer (Long. Beach. Calif).*, vol. 45, no. 5, pp. 34–41, May 2012.
20. T. Kubitza, S. Clinch, N. Davies, and M. Langheinrich, "Using mobile devices to personalize pervasive displays," *ACM SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 16, no. 4, pp. 26–27, Feb. 2012.
21. N. Memarovic and M. Langheinrich, "Beyond Web 2 . 0?: Challenges in Personalizing for Networked Public Display Environments," in *Pervasive Personalisation Workshop at Pervasive 2010*.
22. C. Taivan, J. M. Andrade, R. José, B. Silva, H. Pinto, and A. N. Ribeiro, "Development Challenges in Web Apps for Public Displays," in *7th International Conference on Ubiquitous Computing & Ambient Intelligence, UCAmI 13*, 2013.
23. C. Taivan, R. José, and B. Silva, "Understanding the Use of Web Technologies for Applications in Open Display Networks," in *PD-Apps workshop at PerCom'14*, 2014.
24. R. José, H. Pinto, B. Silva, and A. Melro, "Pins and Posters?: Paradigms for Content Publication on Situated Displays," *IEEE Comput. Graph. Appl.*, vol. 33, pp. 64–72, 2013.
25. S. R. Klemmer, J. Li, J. Lin, and J. A. Landay, "Papier-Mâché: Toolkit Support for Tangible Input," in *Proceedings of the 2004 conference on Human factors in computing systems - CHI '04*, 2004, pp. 399–406.
26. J. Heer, S. K. Card, and J. A. Landay, "prefuse: a toolkit for interactive information visualization," in *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '05*, 2005, p. 421.
27. W3C, "Offline Web applications — HTML5." [Online]. Available: http://www.w3.org/TR/2011/WD-html5-20110525/offline.html. [Accessed: 08-Jul-2014].
28. W3C, "Web Storage." [Online]. Available: http://dev.w3.org/html5/webstorage/. [Accessed: 08-Jul-2014].
29. W3C, "Indexed Database API." [Online]. Available: http://www.w3.org/TR/IndexedDB/. [Accessed: 08-Jul-2014].
30. W3C, "HTML5 Web Messaging." [Online]. Available: http://www.w3.org/TR/webmessaging/. [Accessed: 08-Jul-2014].